Tutorial Builder System

George Cherian – 6408

Neil Chettiar – 6409

Omkar Narkar – 6434

Fr. Conceciao Rodrigues College of Engineering

B.E. Computers

Mumbai University

# Introduction

As it stands today every time we need to gather information about any topic we need to look for data across multiple sites, read through pages and pages of data which are scattered over a host of different websites. Even sites which have data in a single place only deal with giving information which is not easy to use or understand and often prone to human errors during editing.

The Tutorial Builder System that we propose will be used to gather data from different places over the internet and compile all of this data into a set of chapters and data which is easy to understand for the user.

## Objective:

The Tutorial Builder System will take data from multiple sources and combine them into a bank with the data following which the system will read the data from the various sources and strip it down, clean the data up and merge it with data from other sources to get a final version of the data which is user friendly and easy to understand. With this we can eliminate the need to go through multiple web pages to understand or learn a topic with all the information being consolidated in a single space.

## Problem statement:

The Tutorial Builder System is a software based system which will use data collection algorithms, data ranking algorithms, word processing and data merging algorithms to find the data regarding a certain topic and then sort through the data on the topic, combine all of the data in a easy to understand and intuitive manner.

Once the data is put into a database we will proceed to take a apart the data and then convert it back into a properly formed tutorial.

The user will give an input to the system in the form of the topic for which they want the tutorial and preferably the stream under which the tutorial comes followed by which our system will search through the databank for the appropriate tutorial and give it to the user to read the data from the databank.

The back end of the system will compile the data for the databank, by reading the data from various sources and then processing them appropriately and storing the data within the databank.

## Scope:

The scope of this project would be the gathering of data from the internet and checking its relevance, followed by the ranking of gathered data and storing into a database. The data stored in the database will then have to be appropriately processed and combined with other data from various sources and then put into a databank from where the GUI will read the data and present it to the user.

## Applications:

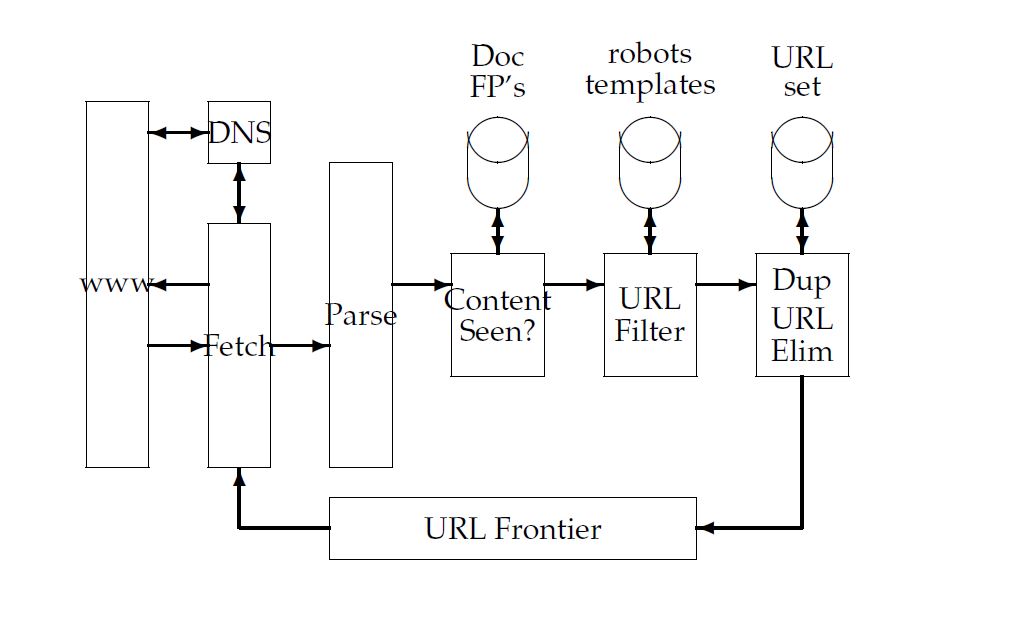
The Tutorial Builder System will have a wide variety of applications in any and all fields. When successfully implemented the system can make learning any subject easier. Instead of the user going to the trouble of accessing multiple sites the system will provide the user with a one stop destination for information about any topic needed by the user.

Currently we plan to implement this system from a computer science standpoint taking into account mostly topics which are within the domain of computer science.

# Literature Survey

## Web Crawler

* Architecture



The simple scheme outlined above for crawling demands several modules that fit together:

1. The URL frontier, containing URLs yet to be fetched in the current crawl (in the case of continuous crawling, a URL may have been fetched previously but is back in the frontier for re-fetching).
2. A *DNS resolution* module that determines the web server from which to fetch the page specified by a URL.
3. A fetch module that uses the http protocol to retrieve the web page at a URL.
4. A parsing module that extracts the text and set of links from a fetched web page.
5. A duplicate elimination module that determines whether an extracted link is already in the URL frontier or has recently been fetched.

A crawler thread begins by taking a URL from the frontier and fetching the web page at that URL, generally using the http protocol. The fetched page is then written into a temporary store, where a number of operations are performed on it. Next, the page is parsed and the text as well as the links in it are extracted. The text (with any tag information – e.g., terms in boldface) is passed on to the indexer. Link information including anchor text is also passed on to the indexer for use in ranking. In addition, each extracted link goes through a series of tests to determine whether the link should be added to the URL frontier.

First, the thread tests whether a web page with the same content has already been seen at another URL. The simplest implementation for this would use a simple fingerprint such as a checksum (placed in a store labeled "Doc FP’s" in the figure). A more sophisticated test would use shingles instead of fingerprints.

Next, a *URL filter* is used to determine whether the extracted URL should be excluded from the frontier based on one of several tests. For instance, the crawl may seek to exclude certain domains (say, all .com URLs) – in this case the test would simply filter out the URL if it were from the .com domain.

A similar test could be inclusive rather than exclusive. Many hosts on the Web place certain portions of their websites off-limits to crawling, under a standard known as the *Robots Exclusion Protocol*.

This is done by placing a file with the name robots.txt at the root of the URL hierarchy at the site.

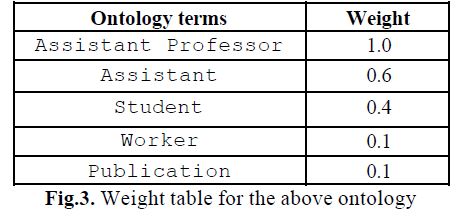
Finally, the URL is checked for duplicate elimination: if the URL is already in the frontier or (in the case of a non-continuous crawl) already crawled, we do not add it to the frontier. When the URL is added to the frontier, it is assigned a priority based on which it is eventually removed from the frontier for fetching. Certain housekeeping tasks are typically performed by a dedicated thread. This thread is generally quiescent except that it wakes up once every few seconds to log crawl progress statistics (URLs crawled, frontier size, etc.), decide whether to terminate the crawl, or (once every few hours of crawling) checkpoint the crawl. In checkpointing, a snapshot of the crawler’s state (say, the URL frontier) is committed to disk. In the event of a catastrophic crawler failure, the crawl is restarted from the most recent checkpoint.

* Additional Features

**Relevance Calculation**

In this section we describe our own algorithm depending on which we calculate relevancy of a Web page on a specific domain.

**Weight Table:** We want to add some weights to each term in the ontology. The strategy of assigning weights is that, the more specific term will have more weight on it. And the terms which are common to more than one domain have less weight. The sample Weight table for some terms of a given ontology of the table shown below:

****

**Relevance calculation algorithm.** In this section we design an algorithm how relevance score of a Web page is calculated.

**INPUT:** A Web page (P), a weight table.

**OUTPUT:** The relevance score of the Web page (P).

**Step1** Initialize the relevance score of the Web page (P) to 0. RELEVANCE\_P=0.

**Step2** Select first term (T) and corresponding weight (W) from the weight table.

**Step3** Calculate how many times the term (T) occurs in the Web page P. Let the number of occurrence is calculated in COUNT.

**Step4** Multiply the number of occurrence calculated at step3 with the weight W. Let call this TERM\_WEIGHT. And TERM\_WEIGHT=COUNT

\* W.

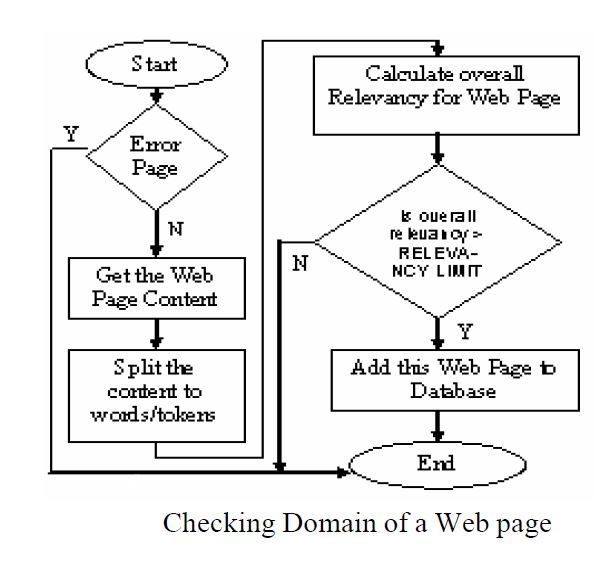
**Step5** Add this term weight to RELEVANCE\_P. So new RELEVANCE\_P will be, RELEVANCE\_P = RELEVANCE\_P + TERM\_WEIGHT.

**Step6** Select the next term and weight from weight table and go to step3, until all the terms in the weight table are visited.

**Step7** End.

**Checking Domain of a Web page**

Using ontological knowledge we can find relevant

Web pages from the Web.

# Machine Learning

Our approach is, like many other popular content extraction methods, text block based. We blockify training documents and train an SVM classifier based on features extracted from the blocks. For test documents, the same blockify algorithm is applied and the blocks classified as content are extracted to construct the main content.

A. Blockifying

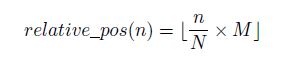
Although many previous works are block-based, they do not document the blockifying process clearly. In our approach, we use HTML parser to traverse the HTML DOM tree and accumulate relevant information such as text and links as traversing, when the beginning or end of a block-level element1 is reached, we output the accumulated text as a text block.

B. Feature Extraction

As we are transforming the content extraction problem into a classification problem, we wish to use features that are strongly indicative whether a text block is content or not. In our approach, we select three types of features: text features, relative position, and id&class token feature. The first type is called text features, which are extracted based on text properties inside a text block. According to the evaluation of block features based on their information gain (Kullback-Leibler-divergence) in [1], for each block we have selected seven most relevant features:

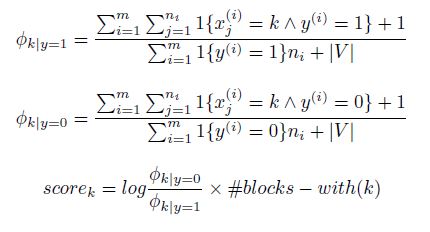
* number of words in this block and the quotient to its previous block
* average sentence length in this block and the quotient to its previous block
* text density in this block and the quotient to its previous block
* link density in this block

The definition of features such as number of words and average sentence length are intuitive and trivial, and the link density is calculated as the ratio of number of words within all <a> tags associated with the text block to the total number of words in the block. As in, we assum that the there are 80 characters in a line, and the text density of a block is defined as the average number of words in one line, which is calculated as the ratio of the total number of words to the total number of lines.

The second type of feature is the relative position of a block in the webpage. To be specific, for a document divided into N blocks, we discretize their position into M relative positions, and the relative position of the nth block is 

The last type of feature is the id & class token feature. In modern HTML documents, the id and class attributes capture the semantic information in the HTML left behind by programmers. For example, tokens such as ad and nav usually indicate that the associated elements are non-content. But of course we don’t want to manually analyze and collect such indicative tokens, which will be not only heuristic-based but also non-efficient. Thus we employed the event-model

Naive Bayes algorithm to learn the top N (which we set to 10) id and class tokens that are the surest indicators of non-content blocks2. We compute the probability and score of k-th token as follows (assuming there are m blocks):



The first two equations are multinomial event model with Laplace smoothing and the third equation is weighted score of k-th token. We select tokens with 10 largest scores3 and add 10 binary features indicating whether a block has the corresponding token or not.

C. SVM Training and Classification

After feature extraction, we label each block as content or non-content based on whether they appear in the gold standard.

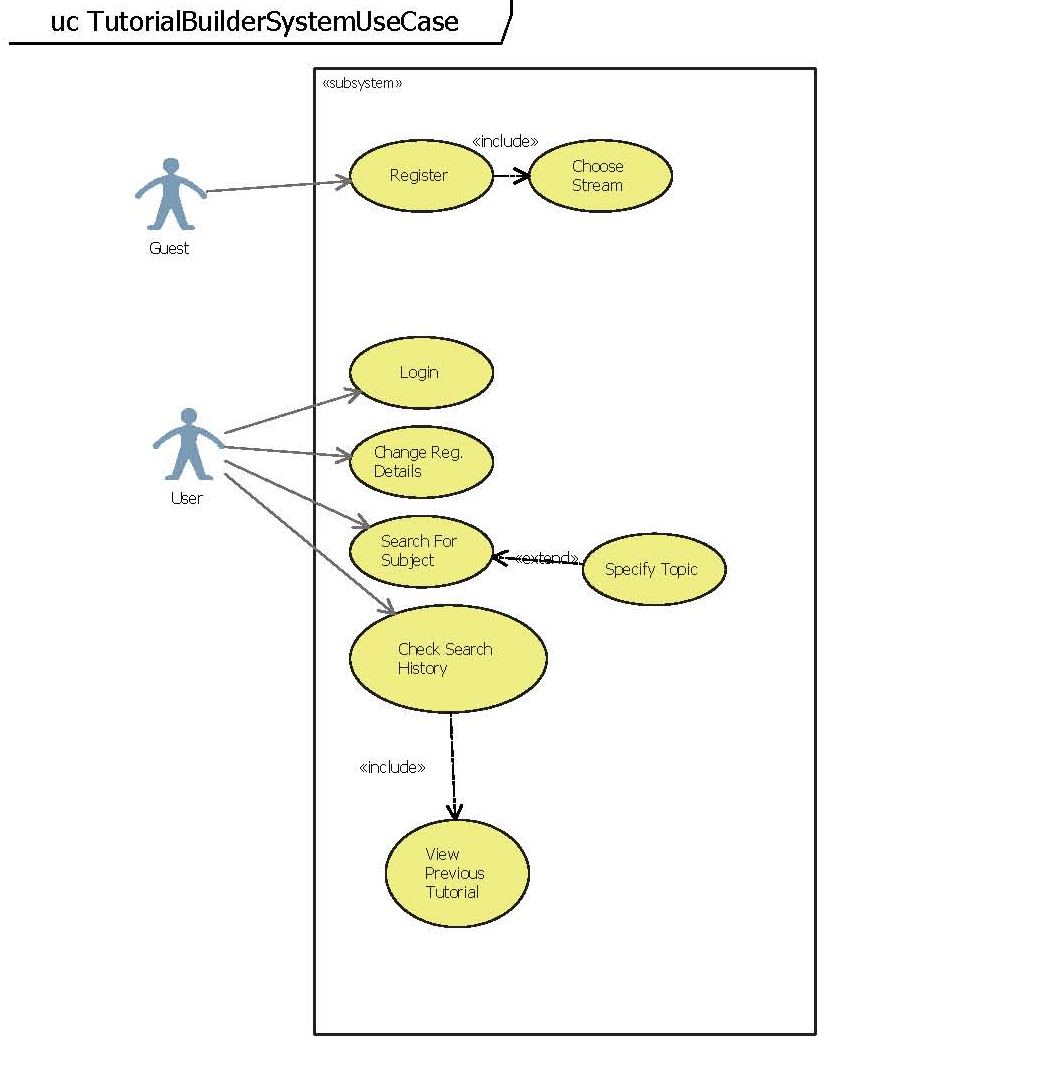
Then we employ SVM to this binary classification problem on the block level with eighteen features as decried in the previous section. Following the suggestions in, we first scale all the attributes’ value to the same range so that the features with a higher absolute value wouldn’t dominate the value of the kernel output. Since we are using large datasets and the dimension of our feature set is relatively small, radial basis function (RBF) would be a good choice as the kernel.

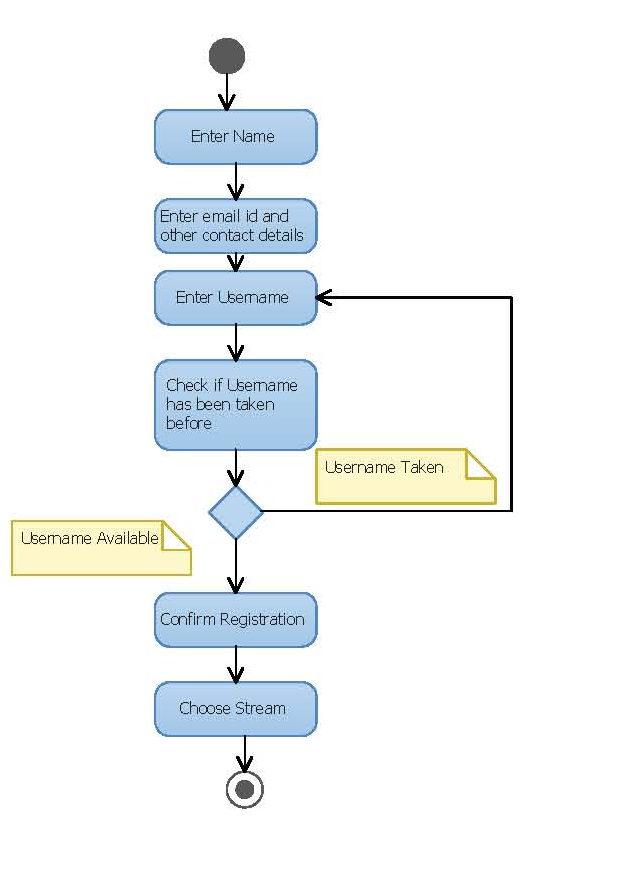
After the SVM classifier is obtained, when a test document is given, we first use the same blockify method to break test document into blocks and then use the classifier to label the blocks as content or non-content. Texts from all the content blocks are then merged to form the extracted content.

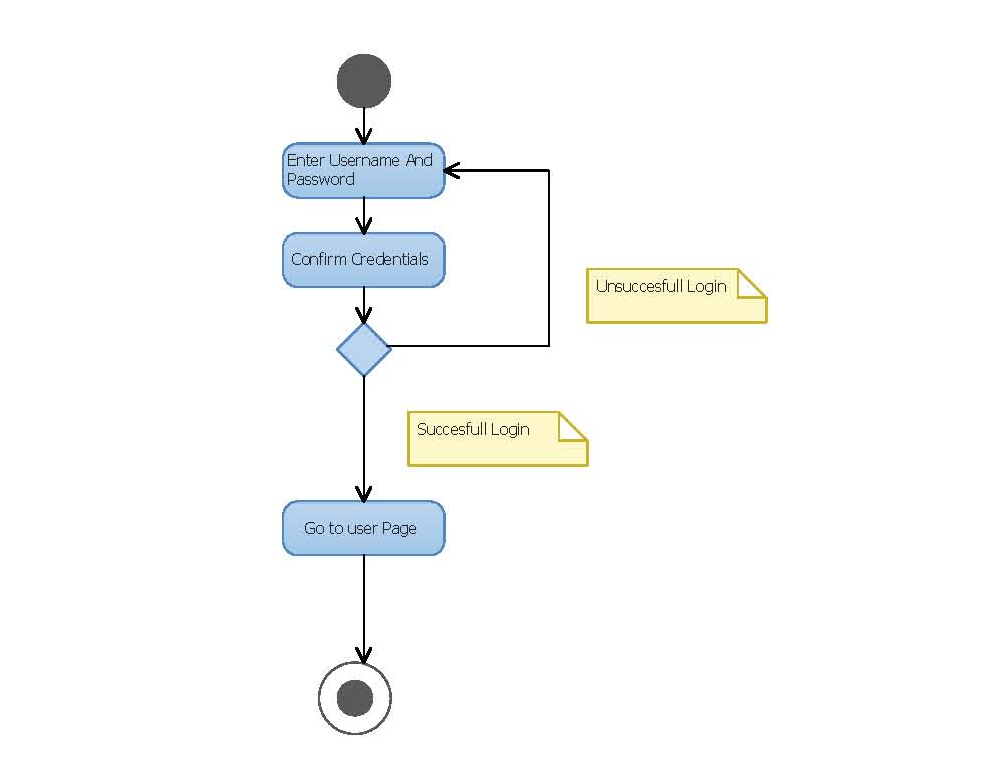
**Data Ranking**

# Proposed System

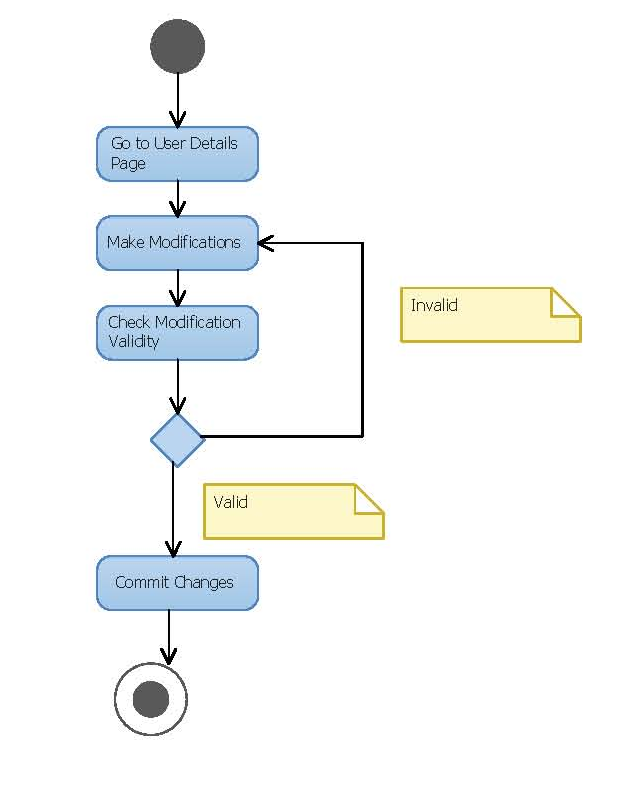
The system that we plan to implement will be directly user input related where the use will enter the input for the subject they would like to search for and then choose a stream into which the search will go, following this they can choose a particular sub topic or choose from a list of topics generated by the system.

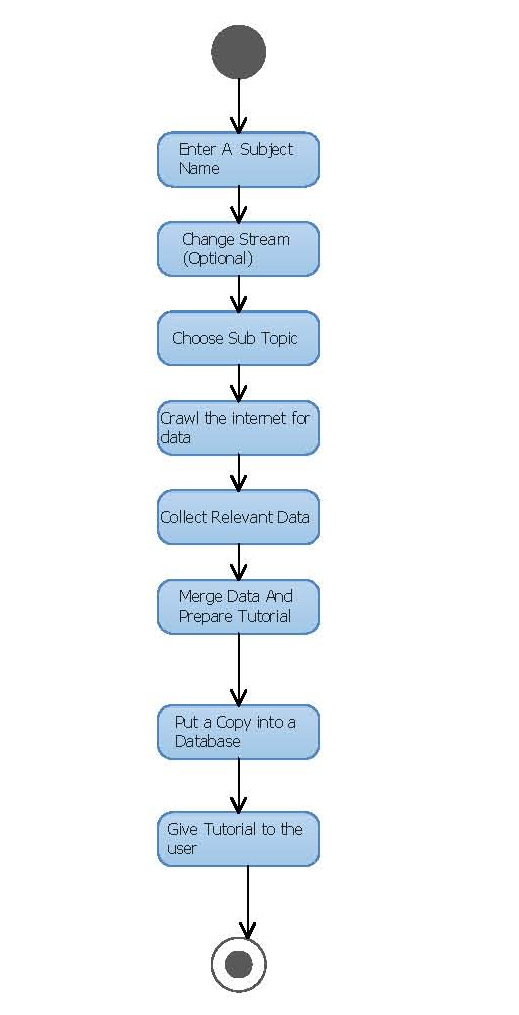
System Use Case

Register Activity Diagram

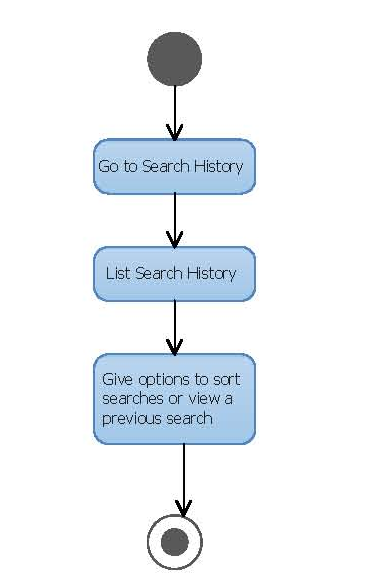
Login Activity Diagram

Change Registration Details Activity Diagram

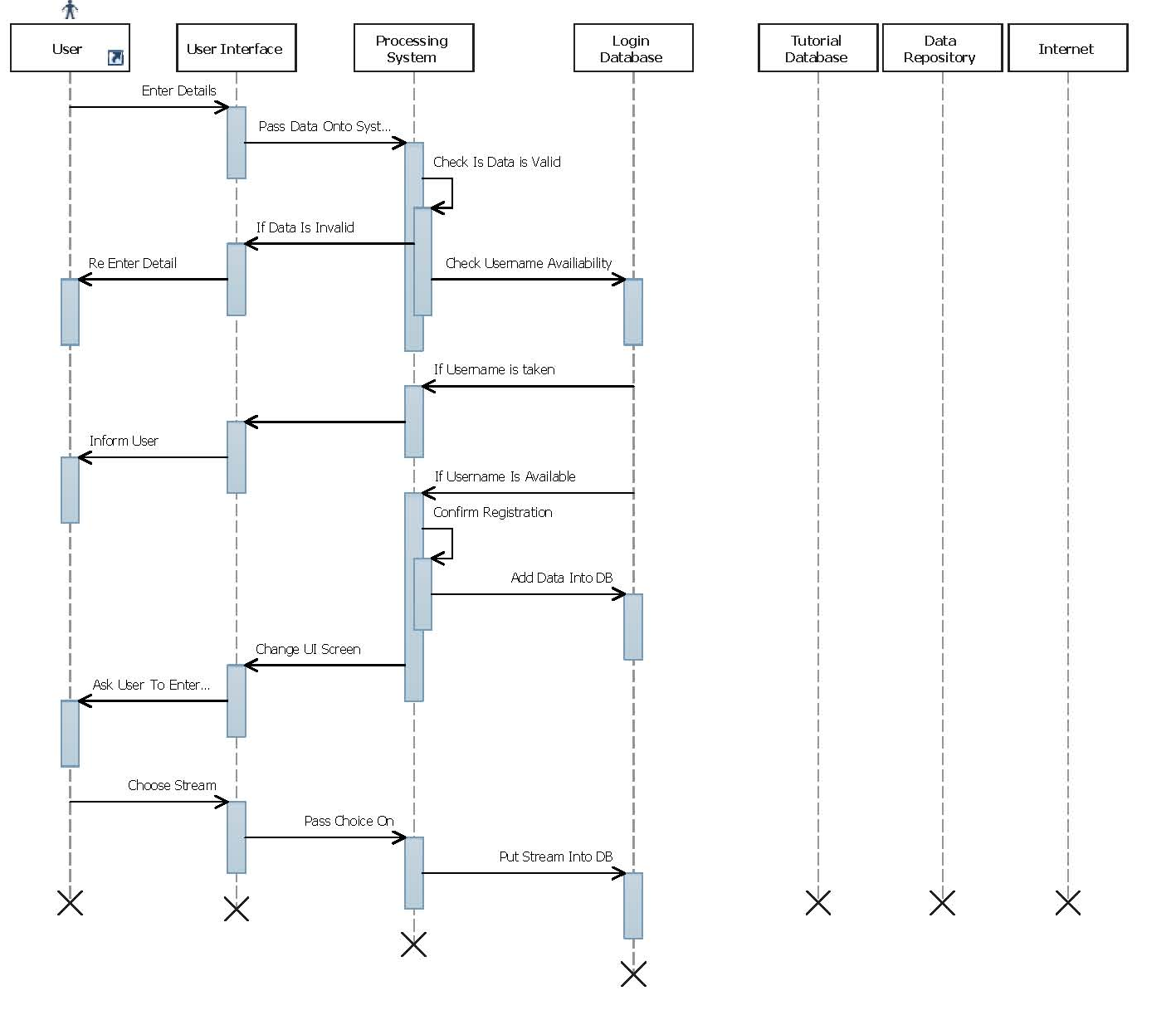


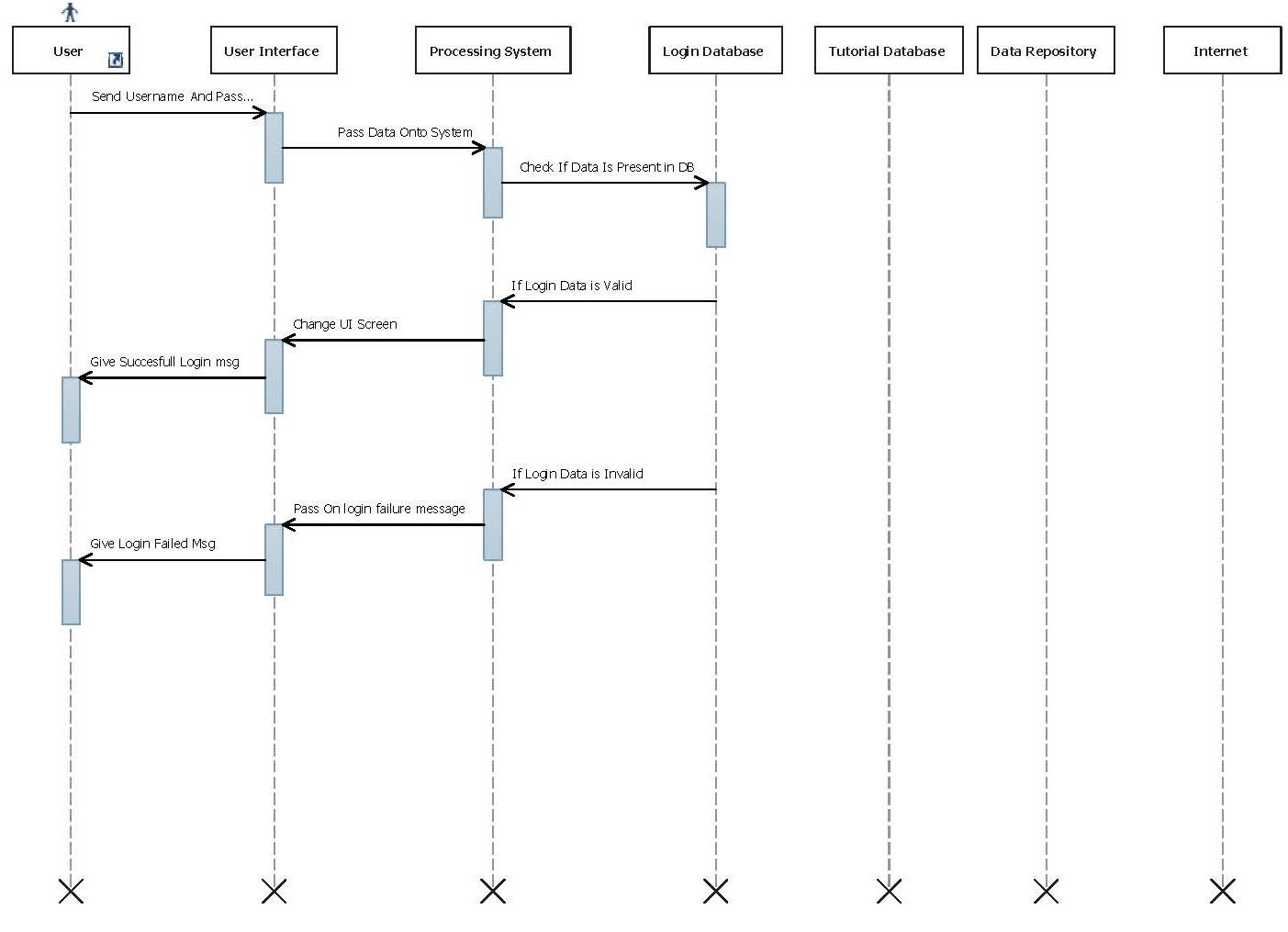
New Search Activity Diagram

View Search History Activity Diagram

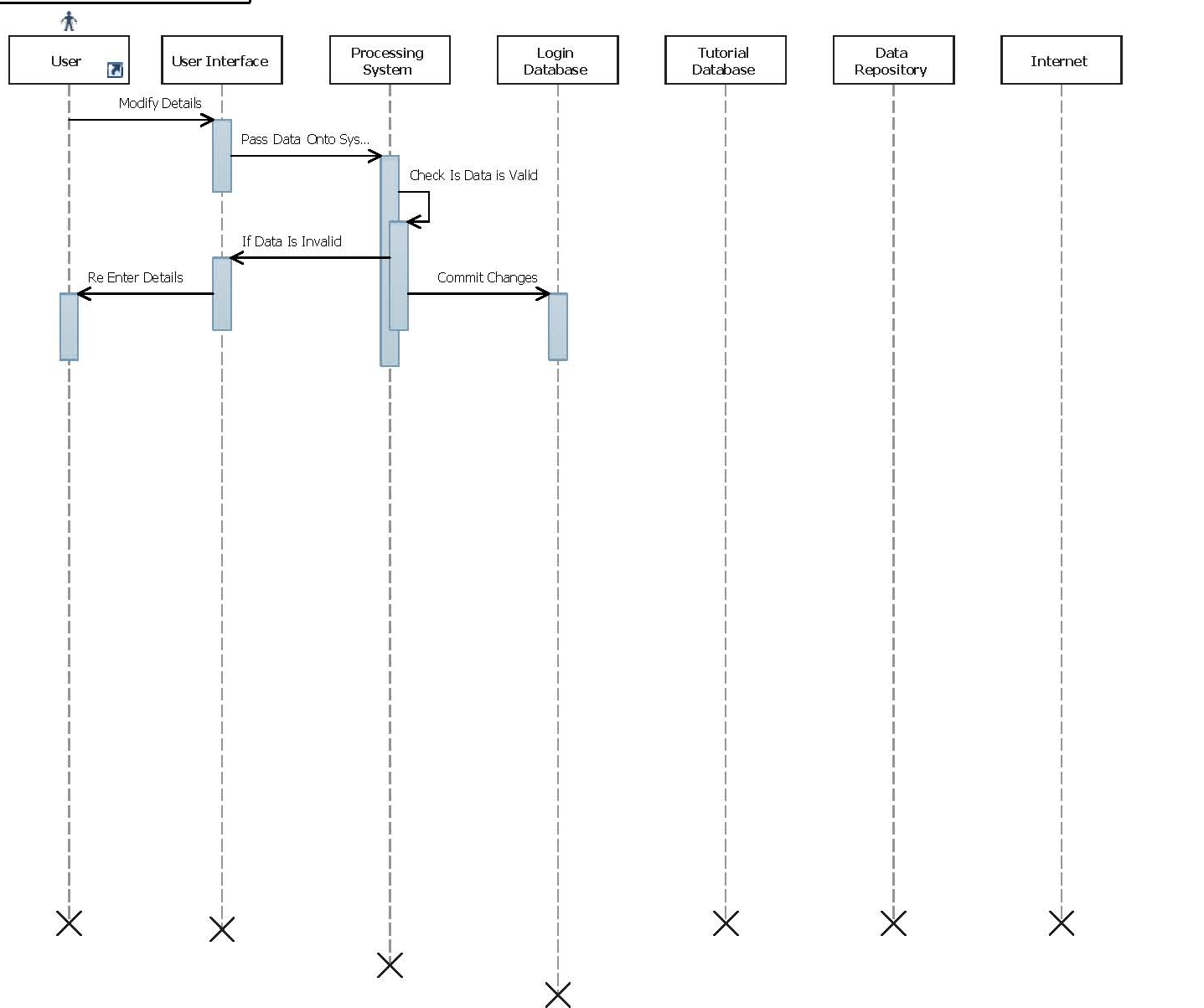


Register Sequence Diagram

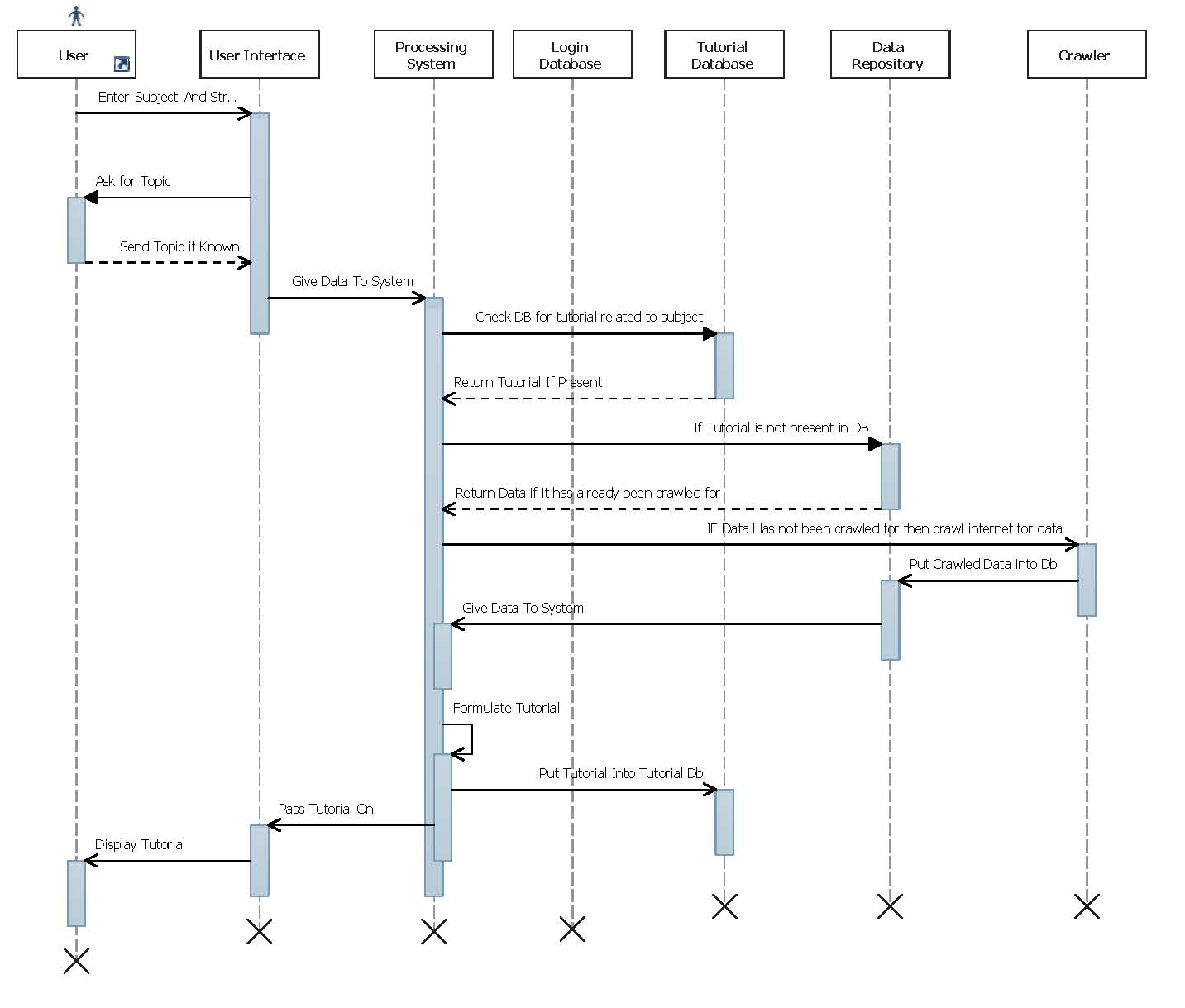


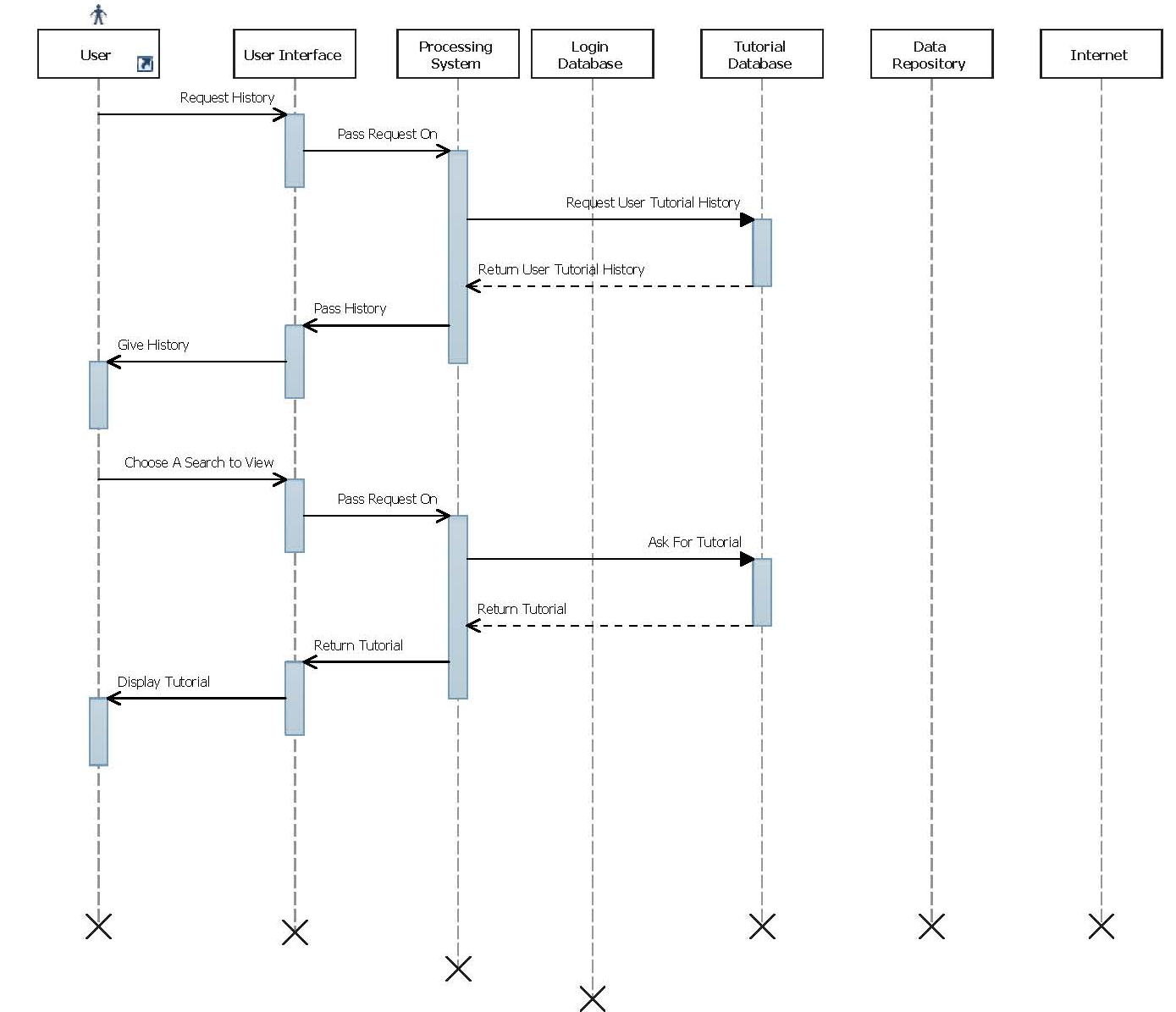
Login Sequence Diagram

Change Registration Details Sequence Diagram



New Search Sequence Diagram



View Previous Search Activity Diagram

# Methodology

We plan to set up a User Interface as either a web application or an Android app, The user will first register themselves onto the system giving all the necessary details and their main stream of interest. After Registering the user will be able to change any of the details entered except for their username. The user can then login any time using the username and password they provided when registering. To get a tutorial the user will enter the subject name on which they want the tutorial, the stream will be automatically set as the stream they had chosen while registration but if they so choose the user can change the stream for any particular search.

The User can then either provide a particular topic on which he wants a tutorial or let the system look for data and then provide a list of topics. Either way a tutorial search request is submitted to the system, The System then proceeds to use crawlers to go through the internet and pick up stream relevant and subject relevant data and put this data into a data base of all the data picked up from the internet.

Once this data has been collected from the internet the system will then proceed to rank the data according to various algorithms and find out the topics within the given subject. Followed by which the data needed for various topics is taken from various sources and then made into a tutorial. A copy of the tutorial is stored into a tutorial database and then sent to the user.

Depending on whether the user specified a topic or not, either the data needed is returned or a list of topics is returned to the user respectively. The user will have all the data in every topic ready with them from various sources.

The user will also have the option to view their search history and see all the previous searches made and also have the option to view a single previous topic or subject searched for and the data contained within that topic.

# Project Timeline